Partielle Lösungen zur „allgemeinen“ Problematik

Eine grundlegende Einführung

Florian Rittberger

BACHELORARBEIT

eingereicht am
Fachhochschul-Bachelorstudiengang
Software Engineering

an der
Fachhochschule Oberösterreich
in Hagenberg

2025

Betreuer*in: Andreas Johann Scheibenpflug BSc MSc

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt. Die vorliegende, gedruckte Arbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Hagenberg, am 1. Juli 2025

Florian Rittberger

Inhaltsverzeichnis

Vorwort

Kurzfassung

Maximal 1-seitige „Zusammenfassung“ der Arbeit in Deutsch.

Abstract

This should be a 1-page (maximum) “summary” of your work in English.

Kapitel 1

Einleitung

Wenn man automatisiert handeln möchste braucht man eine Stragie. Zum testen braucht man auch etwas. Den Backtester. Arbeit wird ein Backtster entwickelt. Der clientseitig ist. Weniger Sicherheitslücken. Doch wie ist der Backtester am schnellsten. Deswegen rechenintenveste Teil ausgelagert. Die Strategien. Diese mit neuen Web Technologien wie WASM im Vergleich zu reinem JS. Nur diese 1 Seite

1.1 Kapitelübersicht

Kapitel 2

Grundlegende Konzepte und Technologien

Dieses Kapitel behandelt die theoretischen und technischen Grundlagen des Backtestings. Dabei werden zentrale Konzepte, mögliche Verzerrungen, bewährte Handelsstrategien, bestehende Backtester sowie weitere Technologien wie WebAssembly erläutert.

2.1 Backtesting

Backtesting bietet die Möglichkeit den Erfolg einer Investitionsstrategie an historischen Finanzdaten zu testen. Die Erfolgsquote dieser Strategie lässt sich somit einschätzen, ohne dafür spekulativ Geld in aktuelle Märkte investieren zu müssen. Dabei wird eine Strategie zu konkreten Zeitpunkten in gewissen Intervallen befragt, ob in diesem Moment gekauft, verkauft oder gehalten werden soll. Die Strategie erhält aber für die Berechnung der Entscheidung nur Informationen, die zu diesem Zeitpunkt auch tatsächlich verfügbar gewesen wären. Zukünftige Informationen, die zu diesem Zeitpunkt noch nicht bekannt waren, dürfen nicht in die Entscheidungsfindung mit einfließen, obwohl diese verfügbar sind. [6]

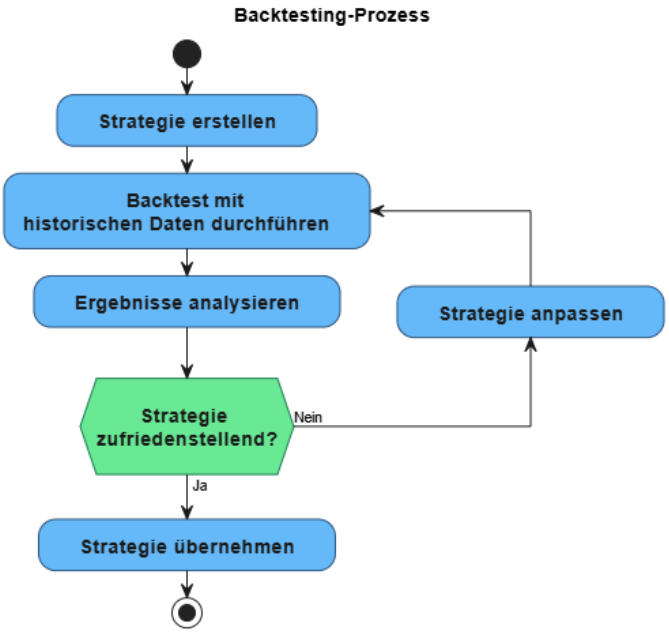


Abbildung 2.1: Ablauf eines Backtestingsprozesses

In der Abbildung 2.1 ist der Ablauf eines Backtestingprozesses dargestellt. Zuerst wird eine Investmentstrategie erstellt, die getestet werden soll. Wie die Strategien implementiert sein muss hängt von dem verwendeten Backtester ab. Daraufhin wird die Strategie auf den historischen Daten im Backtester angewendet. Die Ergebnisse dieses Tests müssen anschließend analysiert werden, um die Qualität der Strategie bewerten zu können. Dazu können verschiedenste Kennzahlen verwendet werden, wie Beispielsweise der prozentuelle Gesamtgewinn oder der Sharpe-Ratio, welcher das Verhältnis von Rendite zu Risiko misst.
Der Sharpe-Ratio wird wie folgt berechnet:

$$\textit{Sharpe-Ratio} = \frac{R_p - R_f}{\sigma_p}$$

- R_p = Rendite des Portfolios
- R_f = Risikofreier Zinssatz
- σ_p = Standardabweichung der Überschussrendite des Portfolios

Falls die Ergebnisse den Erwartungen entsprechen, ist der Backtestingprozess abgeschlossen. Andernfalls sollte die Strategie angepasst werden und der Backtest erneut ausgeführt werden. Dabei sollte der Prozess jedoch nicht zu oft wiederholt werden, da sonst die Gefahr besteht, dass die Strategie zu stark an die historischen Daten angepasst wird und unter realen Bedingungen nicht mehr den Erwartungen entspricht. [24] [4]
Nur weil eine Strategie erfolgreich im Backtesting war, heißt dies aber nicht, dass diese auch in der Zukunft erfolgreich sein wird. Ein möglicher Grund hierfür könnten Verzerrungen (Biase) in der Strategie oder bei den Testdaten sein, die während des Backtests

einbezogen wurden. Das Ergebnis erscheint dadurch im Test überzeugend, in der Praxis aber wird die Strategie den Erwartungen voraussichtlich nicht gerecht. [5]
Desweitern kann im Backtester ein verfälschtes Ergebnis entstehen, wenn während des Testprozesses keine Gebühren oder Steuern berücksichtig werden, die in der Praxis anfallen würden. Strategien mit einer geringen Anzahl an Transaktionen wirken im Backtest weniger profitabel als Strategien mit hoher Handelsfrequenz. In der Realität können sie jedoch aufgrund niedrigerer Transaktionskosten und geringerer Slippage oft bessere Ergebnisse erzielen. [9]

2.2 Biase in Backtesting

Durch falsches Anwenden eines Backtesters können verschiedene Biase in die Ergebnisse mit einfließen und diese dadurch verfälschen. Nachfolgend werden vier bekannte Biase beschrieben und wie diese vermieden werden können.

2.2.1 Look-Ahead Bias

Investitionsstrategien, die Informationen verwenden, die sie zum Zeitpunkt der Entscheidung noch nicht haben konnten, sind vom Look-Ahead Bias betroffen. Dieser verfälscht das Backtestergebnis. Wenn die Strategie die Aktienkurse der folgenden Jahre in die Entscheidungsfindung mit einfließen lässt, kann diese Aktien, bei denen ein Kursanstieg erwartet wird, kaufen und Aktien, bei denen ein Kursrückgang erwartet wird, verkaufen. Somit wird die Strategie im Backtest einen Gewinn erzielen, aber bei realen Bedingungen, bei denen nicht in die Zukunft geblickt werden kann, versagen. Um dies zu verhindern, sollten ausschließlich Informationen berücksichtigt werden, die zum Zeitpunkt der Entscheidung tatsächlich vorlagen. [18]

2.2.2 Overfitting Bias

Der Overfitting Bias entsteht, wenn die erstellte Handlungsstrategie zu stark an die ausgewählten historischen Daten angepasst wurde, um bessere Ergebnisse zu erzielen. Dadurch wird die Strategie nur auf diese speziellen Datensets gute Ergebnisse liefern, in der Praxis aber wird dadurch voraussichtlich die Erwartung nicht erfüllt. Ein Beispiel für eine angepasste Strategie sind statisch definierte Kauf- und Verkaufsgrenzen. Um dies zu vermeiden, sollte die Strategie auf mehreren verschiedenen Datensets getestet werden. Ebenfalls sollte zur Erstellung der Strategie ein anderer Teil der historischen Daten verwendet werden als zum Testen der Strategie. [6]

2.2.3 Survivorship Bias

Der Survivorship Bias entsteht, wenn in den historischen Daten nur Firmen enthalten sind, die während des beobachteten Zeitraums weiterhin bestanden haben. Firmen, die in der Vergangenheit in Konkurs gegangen sind oder aus anderen Gründen nicht mehr existieren, sind in den Daten deswegen nicht enthalten. Somit wird die Strategie im Backtest nur auf Firmen angewandt, die es geschafft, haben zu überleben und dabei wird nicht getestet, ob die Strategie auch bei Firmen erfolgreich wäre die in der Vergangenheit gescheitert sind. Um Verzerrungen zu vermeiden, sollten historische Daten einbezogen

werden, die auch Unternehmen berücksichtigen, die im betrachteten Zeitraum insolvent geworden sind. [13] [14]

2.2.4 Data-Snooping Bias

Ein Testergebnis fällt in den Data-Snooping Bias, wenn für einen Test eine zu kleine Auswahl an unterschiedlichen Kursentwicklungen in den Testdatensets verwendet wurde. Daraus entsteht das Problem, dass die Strategie nur auf die in den Datensets vorkommenden Kursentwicklungen optimiert wurde, ohne auf andere Kursentwicklungen Rücksicht zu nehmen. Wenn alle verwendeten Kurse beispielsweise nur von Firmen stammen, die alle in dem betrachteten Zeitraum einen wirtschaftlichen Aufschwung hatten, wird die Strategie auch nur in ähnlichen Szenarien gute Ergebnisse liefern. Somit wird die Strategie in anderen Marktsituationen möglicherweise nicht die Erwartungen erfüllen. Um dies zu vermeiden, sollten Datensätze gewählt werden, die unterschiedliche Aufschwünge und Krisen aufzeigen, um verschiedene Marktsituationen abzudecken. [42] [35]

2.3 State of the Art Backtester

Es gibt bereits zahlreiche Backtester von den verschiedensten Anbietern. Ein Großteil dieser Systeme ist jedoch kostenpflichtig und laufen nicht lokal, sondern nur auf den Servern der einzelnen Anbieter. [60] [59] [50] Somit werden auch selbst entwickelte Strategien an diese Dienstleister weitergegeben. Dritte könnten potenziell die zugrunde liegenden Konzepte der Ansätze übernehmen und für eigene Zwecke adaptieren. Dies ist ein Sicherheitsrisiko für nicht veröffentlichte Strategien. Darüber hinaus ist bei solchen Anbietern häufig auch nicht transparent, wie die Backtester intern implementiert sind. Im Rahmen dieser Arbeit werden daher nur lokale Open-Source Backtester analysiert. Dadurch lassen sich die genannten Probleme vermeiden und die Backtester im Detail untersuchen und besser miteinander vergleichen. Nachfolgend werden drei bekannte open-source Backtester beschrieben und deren Aufbau und Funktionsweise erläutert.

2.3.1 Backtrader

Backtrader ist ein in Python implementierter Backtester. Alle Komponenten des Backtesters sind in einzelne Klassen aufgeteilt. Eine Klasse für Finanzdaten, Backtesteinstellungen und Startparameter, eine für die Strategie, eine für einen Kauf oder Verkauf und eine Klasse für alle errechneten Ergebnisse. Um selbst Strategien zu implementieren, muss eine Strategieklasse erstellt werden. Die eigene Strategieklasse muss von einer Basisklasse erben und gewisse Funktionalitäten implementieren. Von der Art der Strategie hängt ab, ob alle oder nur ein Teil der Funktionen implementiert werden muss. Mit geerbten Funktionen können beispielsweise Kauf- und Verkaufssignale an den Backtester gesendet werden.
Backtrader bietet keine graphische Benutzeroberfläche, um Backtests zu starten, durchzuführen oder zu analysieren. Der Backtest wird durchgeführt, indem die selbst erstellte Python-Datei ausgeführt wird. Als Datenquelle können CSV-Dateien oder Daten von Online-Quellen wie Yahoo Finance verwendet werden. Alle diese Quellen müssen in ei-

nem bestimmen Format vorliegen. Die Resultate des Backtests werden von Backtrader nicht automatisch visualisiert. Stattdessen müssen die Kennzahlen selbstständig aufbereitet und dargestellt werden. Dadurch gestaltet sich die Präsentation aufwändiger, jedoch wird diese dadurch individuell anpassbarer. Zur Veranschaulichung steht eine Hilfsklasse zur Verfügung, die die Darstellung erleichtert und sich zudem durch Ableitung mit eigenen Grafiken erweitern lässt. Ebenfalls vereinfacht die Bibliothek PyFolio die Visualisierung der Ergebnisse. Mit dieser werden aus den Resultaten verschiedene Diagramme und Tabellen mit Kennzahlen erstellt. [58] [39] [54]
Die Performanz von Backtrader ist durch die Geschwindigkeit von Python begrenzt. Backtrader biete keine Möglichkeit die langsamsten Teile der Strategie in einer anderen Form zu implementieren, um die Ausführungsgeschwindigkeit zu beschleunigen.

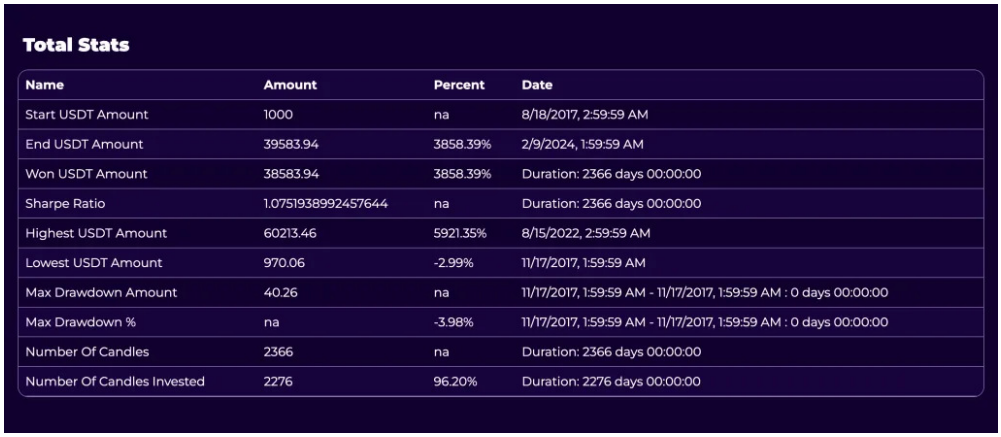
2.3.2 Zipline

Zipline ist ein in Python implementierter Backtester. Dieser wurde ursprünglich von Quantopian entwickelt und wird jetzt von der Community weitergeführt, da 2020 Quantopian den Betrieb eingestellt hat. Die weiterentwickelten Versionen von Zipline haben das Ziel, sich von den ursprünglichen Services von Quantopian zu lösen und die Möglichkeit zu bieten, den Backtester lokal auszuführen. Zwei der bekanntesten Forks sind Zipline Trader und Zipline Reloaded. Den Entwicklern ist es gelungen, sich von den eingestellten Services zu lösen und eigene Funktionalität weiter hinzuzufügen. Beide Forks sind Open-Source und können kostenlos verwendet werden. Ebenfalls ist es möglich beide Systeme über die Kommandozeile zu bedienen, mit Python-Skripten zu steuern oder diese in einem Jupiter Notebook anzuwenden. Zipline Trader geht mehr in die Richtung von Python Skripten, während Zipline Reloaded mehr für die Verwendung in einem Jupiter Notebook oder der Kommandozeile ausgelegt ist.
Damit eigene Strategien verwenden werden können muss eine Funktion implementiert werden, die eine Sammlung an Daten nimmt und Kauf- und Verkaufssignale zurückgibt. Dabei können verschiedenste Hilfsfunktionen und Indikatoren verwendet werden, die in Zipline bereits implementiert sind. Zipline Trader benötigt, um historische Daten zu laden eine Verbindung zu Alpaca oder Interactive Brokers und eine Postgres-Datenbank. Zipline Reloaded hat standartmäßig Kurse für gewisse Aktien vorinstalliert und kann auch Daten von NASDAQ im pickle Format über die API nachladen.
Die Ergebnisse eines Backtests werden in beiden Systemen automatisch generiert aber müssen manuell visualisiert werden. Dafür können verschiedene Bibliotheken wie Matplotlib, PyFolio oder Quantstats verwendet werden. [55] [29] [57] [56] [30]

2.3.3 BacktestJS

BacktestJS ist ein in TypeScript und JavaScript geschriebener Backtester. Dieser kann nur aus der Kommandozeile heraus verwendet werden und hat keine grafische Benutzeroberfläche. Die entstandenen Ergebnisse liegen als HTML-Dateien vor und können in einem Webbrowser betrachtet werden. In diesem Backtester ist es auch möglich eigene Strategien testen zu lassen. Dazu muss eine Strategie in JavaScript oder TypeScript implementiert werden die gewisse Bedingungen erfüllt. Diese Regeln bestimmen wie die Funktion zu heißen hat, welche Parameter diese annimmt, welche Imports nötig sind und wo diese sich im Dateisystem zu befinden hat. Ebenfalls stellt BacktestJS die Mög-

lichkeit zur Verfügung sich historische Finanzdaten in eine lokale Datenbank zu laden und diese als CSV-Datei zu exportieren. Diese Daten stammen von Binance und sind kostenlos verfügbar. Andere CSV-Dateien in diesem Format können auch direkt in die Datenbank mit BacktestJS geladen werden. Damit ist es möglich seine Strategien auf verschiedene Kurse zu testen. [51] [8]



Total Stats

Name	Amount	Percent	Date
Start USDT Amount	1000	na	8/18/2017, 2:59:59 AM
End USDT Amount	39583.94	3858.39%	2/9/2024, 1:59:59 AM
Won USDT Amount	38583.94	3858.39%	Duration: 2366 days 00:00:00
Sharpe Ratio	1.0751938992457644	na	Duration: 2366 days 00:00:00
Highest USDT Amount	60213.46	5921.35%	8/15/2022, 2:59:59 AM
Lowest USDT Amount	970.06	-2.99%	11/17/2017, 1:59:59 AM
Max Drawdown Amount	40.26	na	11/17/2017, 1:59:59 AM - 11/17/2017, 1:59:59 AM : 0 days 00:00:00
Max Drawdown %	na	-3.98%	11/17/2017, 1:59:59 AM - 11/17/2017, 1:59:59 AM : 0 days 00:00:00
Number Of Candles	2366	na	Duration: 2366 days 00:00:00
Number Of Candles Invested	2276	96.20%	Duration: 2276 days 00:00:00

Abbildung 2.2: Beispielergebnis eines Backtests mit BacktestJS [7]

Ein Beispiel für ein Ergebnis eines Backtests mit BacktestJS ist in der Abbildung 2.2 zu sehen. In dieser Statistik werden verschiedene Kennzahlen wie beispielsweise der Gesamtgewinn, der Startbetrag, der Höchststand der Aktie und der Sharpe-Ratio angezeigt. Es werden nach einem Backtest auch noch weitere Tabellen und Diagramme mit genaueren Informationen generiert und angezeigt.
Die Geschwindigkeit von BacktestJS ist limitiert durch die Ausführungsgeschwindigkeit von JavaScript. Es ist nicht möglich rechenintensive Teile der Strategie in einer anderen schnelleren Programmiersprache wie C oder C++ zu schreiben, um die Backtest-Prozess zu beschleunigen. Dies schränkt den Rahmen für Optimierungen der Laufzeit bei komplexeren Strategien ein.

2.4 Bewährte Ansätze für Handelsstrategien

Es gibt viele bereits etablierte Ansätze für Anlagestrategien. In dieser Arbeit werden keine neuen Algorithmen entwickelt, sondern bestehende Ansätze verwendet, diese in JavaScript und C implementiert, in einem Webbrowser ausgeführt und die Ausführungsgeschwindigkeit getestet. Nachfolgend werden drei bereits bekannte Ideen für Investitionsstrategien beschrieben.

2.4.1 Mittelwertrückkehr mit Bollinger Bänder

Die Idee hinter der Mittelwertrückkehr ist, dass der Kurs einer Aktie immer wieder zu einem längerfristigen Mittelwert zurückkehrt. Dabei werden Abweichungen von dem Mittelwert als Kauf- oder Verkaufssignal verstanden. Es werden die Aktien verkauft,

wenn der Kurs über dem Mittelwert steigt und gekauft, wenn der Kurs unter dem Mittelwert fällt. [36]
Um die Signale zu verstärken und nicht bei den kleinsten Änderungen eine Aktion auszuführen werden häufig Bollinger Bänder verwendet. Diese bestehen aus drei Komponenten.

- Einem gleitenden Durchschnitt des Kurses über einen bestimmten Zeitraum.
- Einem oberen Band, welches eine bestimmte Anzahl an Standardabweichungen über dem Durchschnitt liegt.
- Einem unteren Band, welches eine bestimmte Anzahl an Standardabweichungen unter dem Durchschnitt liegt.

Somit wird ein breiter Mantel über den Aktienkurs gelegt, der die normalen Schwankungen des Kurses umhüllt. Wenn der Kurs das Band nach oben hin durchbricht, wird dies als Verkaufssignal gewertet. Unterschreiten des Bandes wird als Kaufsignal gewertet. [11] [41] [33]

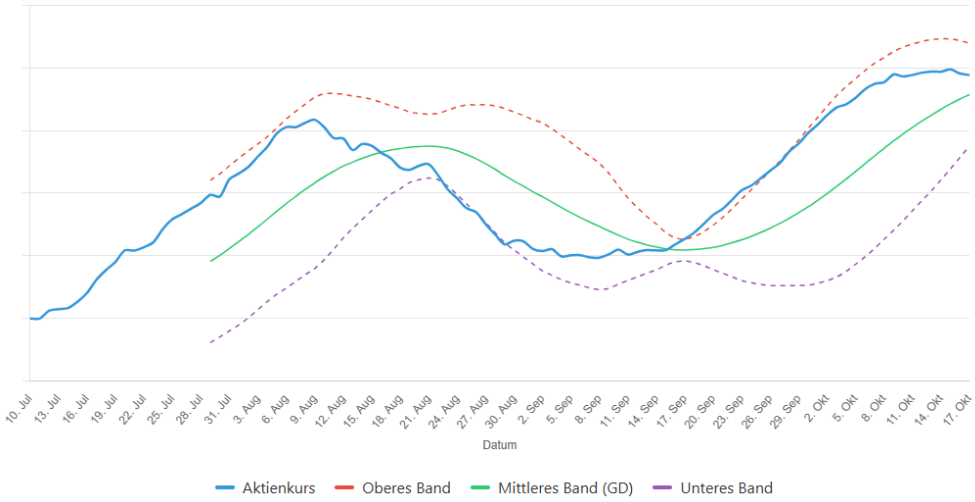


Abbildung 2.3: Beispiel für ein Bollinger Band

In der Abbildung 2.3 ist ein Beispiel für ein Bollinger Band zu sehen. Der blaue Bereich stellt den Kursverlauf der Aktie dar. Die grüne Linie ist der gleitende Durchschnitt, das obere Band ist die rote Linie und das untere Band ist die violette Linie. Am 17. September ist ein Verkaufssignal zu erkennen, da der Kurs das obere Band durchbricht. Am 21. August ist ein Kaufsignal zu erkennen, da der Kurs das untere Band unterschreitet.

Wechselkurse von Währungen dienen als Beispiel dafür, dass Kurse oft zu einem Mittelwert zurückkehren. Diese pendeln häufig um einen Mittelwert und bleiben im Verhältnis zueinander stabil. [21]
Bei Strategien mit Mittelwertrückkehr kann es zu Problemen kommen, wenn der Kurs nicht zum Mittelwert zurückkehrt, sondern sich stark in eine Richtung bewegt. Beispiele dafür sind Firmen, die in Konkurs gehen oder einen starken wirtschaftlichen Aufschwung erleben. [38]

2.4.2 Gleitende Durchschnitts Überscheidungen

Gleitende Durchschnitts Überschneidungen Strategien sind eine der bekanntesten und am häufigsten verwendeten Ansätze im Bereich des automatisierten Handels. Dabei werden zwei gleitende Durchschnitte mit unterschiedlichen langen Zeiträumen errechnet und miteinander verglichen. Wenn der kurzfristige Durchschnitt den langfristigen Durchschnitt von unten nach oben durchbricht, wird dies als Kaufsignal gewertet. Wird der langfristige Durchschnitt aber von oben nach unten durchbrochen, wird dies als Verkaufssignal gewertet. Der Unterschied in den verschiedenen Strategien mit dieser Vorgehensweise liegt in der Wahl der Zeiträume für die beiden gleitenden Durchschnitte und der Mehrgewichtung aktueller Werte. [27] [12]
Eine optimierte gleitende Durchschnitts Überschneidungsstrategie erzielt mehr Rendite als die Strategie eine Aktie zu kaufen und zu warten. Für den S&P 500 Index zeigte die wartende Strategie im Zeitraum von März 2009 bis November 2014 einen niedrigeren Sharpe-Ratio (1,214) als die optimierte gleitende Durchschnittsüberschneidungsstrategie (1,351). Dies bedeutet, dass die optimierte Strategie im Verhältnis zu dem eingegangenen Risiko eine höhere Rendite erzielen konnte. [22]
Ein Problem bei gleitenden Durchschnitts Überschneidungsstrategien ist, dass es zu Fehlsignalen kommen kann, wenn die beiden Durchschnitte sich häufig überschneiden. Dies passiert häufig in Seitwärtsmärkten, bei denen der Kurs keine klare Richtung hat und sich nur wenig verändert. Dabei kann es dazu kommen, dass es sehr viele Kauf- und Verkaufssignale gibt, die einerseits zu hohen Transaktionskosten führen und andererseits bei langsam fallenden Kursen zu Verlusten führen. [3]
Ein weiteres Problem ist, dass diese Strategien sehr von der Auswahl der Zeiträume für die gleitenden Durchschnitte abhängen. Durch eine schlechte Wahl der Zeiträume kann es dazu kommen, dass die Strategie nicht die Erwartungen erfüllt. [20]

2.4.3 Ausbruchsstrategien

Bei Ausbruchsstrategien wird davon ausgegangen, dass der Aktienkurs aus gesetzten Grenzen ausbricht und sich in die Richtung des Ausbruchs weiterbewegt. Zur Bestimmung der Grenzen werden häufig Hoch- und Tiefpunkte in der Kursentwicklung in einem gewissen Zeitrahmen verwendet. Ausbrüche über die Obergrenze werden als Kaufsignal gewertet, während Ausbrüche unter die Untergrenze als Verkaufssignal gewertet werden. [26] [1]
Mit einer optimierten Ausbruchsstrategie kann sogar der S&P 500 Index in einem Backtest übertroffen werden. Im Zeitraum vom Jänner 2016 bis Dezember 2023 konnte der S&P 500 198% Rendite erzielen, während eine optimierte Ausbruchsstrategie 1637% Rendite erzielen konnte. Die Strategie war Risikoreicher als der S&P 500, konnte aber durch das höhere Risiko auch eine deutlich höhere Rendite erzielen. [47]
Ein Problem bei Ausbruchsstrategien ist, dass es zu Fehlsignalen kommen kann. Dies passiert, wenn eine der Grenzen kurzfristig durchbrochen wird, ohne weiter in die durchbrochene Richtung zu laufen. Durch diese Fehlsignale kann es dazu kommen das bei Spitzen gekauft und in Tälern verkauft wird, was zu Verlusten führt. [37]

2.5 WASM

WebAssembly (WASM) ist ein binäres Instruktionsformat für das Web. Es wurde dafür entwickelt, Webanwendungen schneller und effizienter ausführen zu können und diese dabei gleichzeitig portable zu halten. WebAssembly kann in verschiedensten höheren Programmiersprachen wie C, C++, Rust und Go geschrieben werden und wird von allen modernen Webbrowsern unterstützt. Der erstellte Quellcode kann anschließend mit einem Compiler, beispielsweise Emscripten, in Bytecode übersetzt werden und durch Aufrufen mit JavaScript in einem Browser ausgeführt werden. [45]

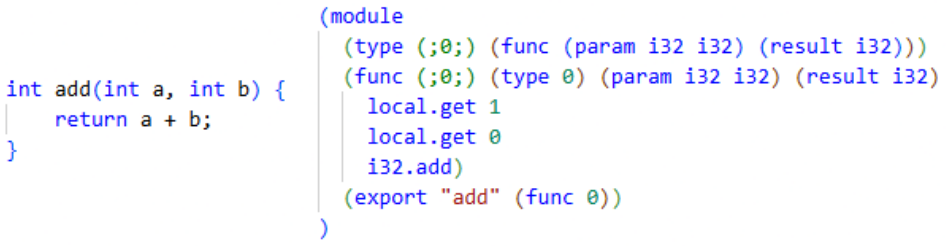


Abbildung 2.4: Vergleich von C Quellcode und generiertem WebAssembly Bytecode

In der Abbildung 2.4 ist ein Beispiel für C Quellcode und der daraus generierte WebAssembly Bytecode zu sehen. In C wird eine Funktion definiert, die zwei ganzzahlige Werte als Parameter annimmt, diese addiert und zurückgibt. Im WebAssembly-Bytecode wird in der zweiten Zeile zunächst der Funktionsprototyp erstellt. Dieser erhält zwei Ganzzahlen als Eingabeparameter und eine Ganzzahl als Rückgabewert. Anschließend erfolgt die Definition der Funktion selbst. In der Funktion werden die beiden lokalen Variablen die als Parameter übergeben wurden, geladen und addiert. Das Ergebnis verbleibt auf dem Stack und stellt dadurch den Rückgabewert der Funktion dar. Das Schlüsselwort export gibt den Namen der Funktion an, damit diese von außen aufgerufen werden kann. Die Metadaten, die durch den Compiler hinzugefügt werden, sind dabei im Bytecode weggelassen. Ebenfalls sind die Präprozessoranweisungen im C Quellcode, um ihn mit Emscripten kompilieren zu können nicht dargestellt.

Da WebAssembly nicht direkt mit dem Browser interagieren kann, gibt es einige Einschränkungen. So hat WebAssembly beispielsweise keinen direkten Zugriff auf das Document Object Model (DOM) eines Webbrowsers. Ebenfalls gibt es keinen automatischen Garbage Collector, was zu Folge hat, dass die Speicher manuell verwaltet werden muss. [45]
Des Weiteren können Funktionen in WebAssembly nur einzelne Werte und keine Verbunde oder Objekte zurückgeben. Damit ist also weniger Sprachumfang als in JavaScript gegeben. Es kann ebenfalls auch nicht auf Umgebungsvariablen beispielsweise dem DOM zugegriffen werden. [**10.1145/3062341.3062363**]

2.5.1 Resourcenverbrauch und Performanz von WASM

Der Vergleich der Ausführungszeiten zeigt, dass WebAssembly in den von [46] getesteten Desktop-Browsern insgesamt schneller ausgeführt wird als JavaScript. Während Firefox JavaScript im Durchschnitt etwas langsamer ausführt als Chrome (1,06× Ausführungszeit), läuft WebAssembly in Firefox deutlich schneller und benötigt nur 0,61× der Ausführungszeit von Chrome. Auch im direkten Vergleich erreicht WebAssembly in Firefox kürzere Laufzeiten als reines JavaScript, aufgrund optimierter Funktionsaufrufe und effizienterer Ausführungspfade. Die absoluten Ausführungszeiten zeigen aber, dass WebAssembly nur in Firefox schneller ist als JavaScript (39,65 ms gegenüber 48,26 ms), während es in Chrome und Edge jeweils langsamer ausgeführt wird als JavaScript. Bei der Laufzeit weist WebAssembly im Vergleich zu JavaScript ebenfalls einen vielfach höheren Speicherverbrauch auf. In allen Desktop-Browsern benötigt WebAssembly deutlich mehr Speicher als JavaScript (3,39x in Chrome, 4,93x in Firefox und 3,44x in Edge). [46]
Auch andere Studien zeigen, wie WebAssembly im richtigen Browser schneller ausgeführt wird als JavaScript. In den Tests von [19] kommt WebAssembly mit einem optimiertem Speicherzugriff und parallelem Datenzugriff beim Multiplizieren einer 102 x 1024 Matrix auf einen 1,64-fachen Geschwindigkeitsvorteil gegenüber einer optimierten JavaScript-Implementierung. [43] zeigt desweitern, dass WebAssembly besonders bei rechenintensiven Aufgaben wie der Bild- und Videobearbeitung oder mathematischen Simulationen eine deutlich bessere Leistung als JavaScript bietet.
Im Vergleich zu nativen Anwenden in C schneidet WebAssembly jedoch schlechter ab. WebAssembly erreicht im Durchschnitt nur etwa 60-70% der Leistung von nativem C-Code. [25] [31]

Kapitel 3

Konzept und Design

In diesem Kapitel werden das Konzept und Design des im Rahmen dieser Arbeit entwickelten Backtesters beschrieben. Zudem wird der Aufbau der Strategien sowie die Herangehensweise bei der Visualisierung der Kennzahlen und Ergebnisse erläutert.

3.1 Architektur des Backtesters

Der entwickelte Backtester besteht aus mehreren Komponenten. Den erhaltenen Daten, der Backtesterlogik und der Visualisierung der Ergebnisse.

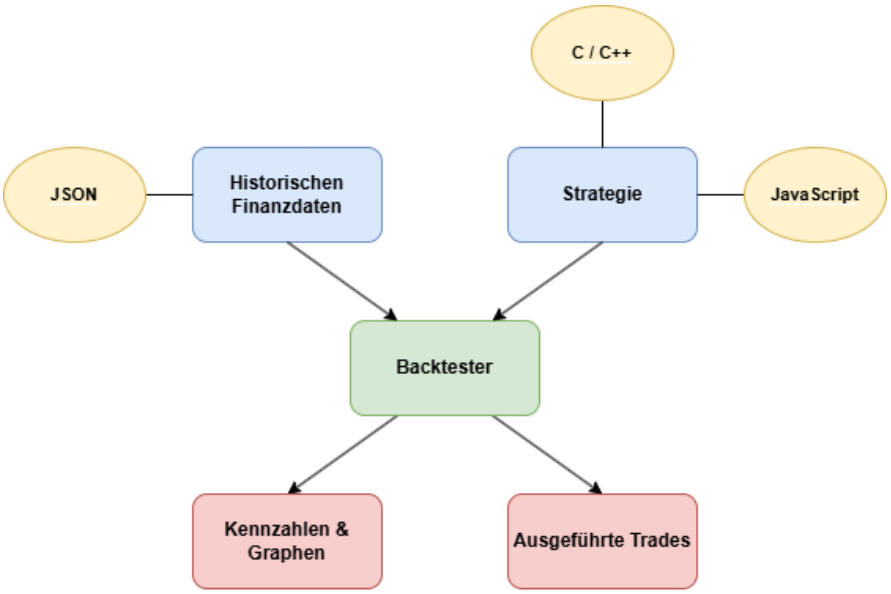


Abbildung 3.1: Struktur des Backtesters

In der Grafik 3.1 ist die Struktur des gesamten Backtesters dargestellt. Zu Beginn werden eingehenden Daten in die Backtesterlogik eingespeist. Einerseits wird eine Historie von Kursdaten im JSON Format übergeben, andererseits wird die zu testende Strategie als WebAssembly Modul oder JavaScript Datei geladen. Als Result entsteht eine

Visualisierung der Backtestergebnisse und die Auflistung der Trades.

3.1.1 Format der historischen Finanzdaten

Die historischen Finanzdaten werden über die Weboberfläche im JSON Format an die Backtesterlogik weitergereicht. Das JSON Format eignet sich für die Übergabe, da es von JavaScript nativ unterstützt wird und leicht zu parsen ist. JSON hat sich als Standartformat für die Datenverwaltung im Web etabliert, da dieses gut Menschen-lesbar ist und performant von Webbrowsern verarbeitet werden kann. Darüber hinaus bringt die Struktur von JSON den Vorteil mit sich, dass komplexe Datenstrukturen damit einfach abgebildet werden können. XML wurde nicht gewählt, da es im Vergleich zu JSON eine umfangreichere Syntax besitzt und dadurch mehr Speicherplatz sowie Rechenzeit beim Parsen benötigt. Des Weiteren ist XML weniger kompakt und wird von modernen Webanwendungen nicht so effizient wie JSON verarbeitet. [48]
Die historischen Daten müssen in einer bestimmten Form vorliegen, um eine korrekte Verarbeitung durch die Backtesterlogik zu ermöglichen. Jeder Datenpunkt ist dabei durch zwei Attribute zu beschreiben: einen Zeitstempel im ISO Format sowie einen zugehörigen Kurswert als Flieskommazahl.

3.1.2 Backtesterlogik

In der Backtesterlogik ist der Hauptteil der Funktionalität des Backtesters umgesetzt. Der Backtestprozess beginnt, nachdem die historischen Finanzdaten und die zu testende Strategie geladen wurden. Dabei wird iterativ über den gesamten Datensatz gegangen und in einem vordefinierten Intervall die Strategie aufgerufen und nach einem Handelssignal befragt. Die Strategie erhält als Eingabeparameter den aktuellen Kurswert, eine festgelegte Anzahl historischer Datenpunkte sowie den zeitlichen Abstand, in dem diese historischen Kurswerte voneinander entfernt erfasst wurden. Die WASM Module erhalten ebenso die Anzahl der übergebenen Kurswerte, damit die Daten korrekt verarbeiten werden können. Je nach zurückgegebenem Signal der Strategie wird danach ein Kauf- oder Verkaufsvorgang simuliert. Dieser Vorgang wird aber nur durchgeführt, wenn bei einem Kaufvorgang genügend Kapital vorhanden ist oder bei einem Verkaufsvorgang genügend Wertpapiere im Portfolio sind. Somit wird verhindert, dass ein negatives Kapital oder ein negativer Bestand an Wertpapieren entsteht. Jede erfolgreiche Transaktion während des Backtestprozesses wird protokolliert, um diese nach vollständiger Berechnung visualisieren zu können. In der Abbildung 3.2 ist dieser Prozess visuell dargestellt.

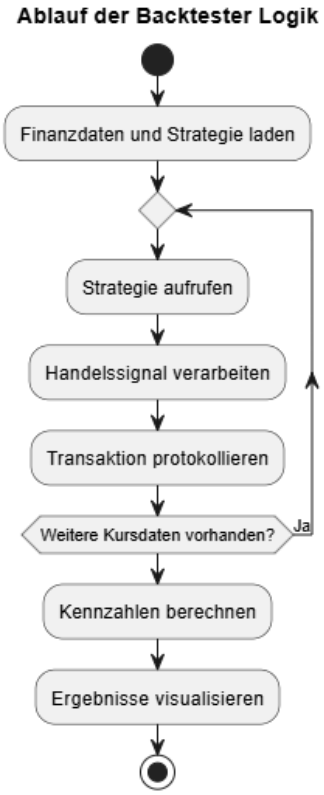


Abbildung 3.2: Backtestingablauf

Ein weiterer Aspekt in der Backtesterlogik ist die Messung der Ausführungsgeschwindigkeit der einzelnen Strategien. Da gleichwertige Strategien mit identer Logik in verschiedenen Programmiersprachen implementiert und getestet werden können, ist es möglich die Ausführungszeit der Strategien zu vergleichen. Dazu wird die Zeit gemessen, die für das Ausführen der Strategie während des Backtestes benötigt wird. Die Zeit startet nach dem Initialisieren des Backtesters und endet, nachdem alle Datenpunkte in den Kursdaten berücksichtigt wurden. Das Laden der historischen Finanzdaten, der Strategie und die Errechnung der Kennzahlen wird dabei nicht mit einbezogen.
Am Ende des Backtestingvorganges werden ebenfalls verschiedene Kennzahlen errechnet, um die Performance der getesteten Strategie zu bewerten. Zu den errechneten Kennzahlen gehören: Endwert des Portfolios, Gesamtrendite, prozentueller Gewinn und der Sharpe Ratio.

3.1.3 Modularisierung der Strategien

Der Backtester ist so gestaltet, dass verschiedene Handelsstrategien modular hochgeladen und getestet werden können. Die Modularisierung ermöglicht es, eigene Strategien zu implementieren und diese in dem Backtester zu verwenden ohne Backtestinglogik verändern zu müssen. Die Strategiemodule müssen dabei nur einen definierten Namen und eine vordefinierte Signatur besitzen, um korrekt aufgerufen werden zu können. Der

Rückgabewert dieser Funktion bestimmt dabei, ob ein Kauf-, Verkaufs- oder Haltesignal ausgegeben wird.

```
int strategy(double *prices, int length, int intervalsBack)

function strategy(prices, intervalsBack)
```

Abbildung 3.3: Funktionsschnittstellen der Strategien in C und JavaScript

In der Abbildung 3.3 sind die Funktionsschnittstellen der Strategien in C und JavaScript dargestellt. Beide Schnittstellen besitzen denselben Funktionsnamen und die logisch gleichen Parameter. Die C-Funktion erhält im Vergleich zur JavaScript Funktion zusätzlich die Anzahl der übergebenen historischen Kurswerte als Parameter, da in C keine dynamischen Arrays unterstützt werden. Beide Funktionen geben einen ganzzahligen Wert zurück, der das Handelssignal repräsentiert.

Das modulare Konzept mit Funktionen erlaubt die Verwendung von Strategien in verschiedenen Programmiersprachen. Der in dieser Arbeit entwickelte Backtester unterstützt Strategien in den Programmiersprachen JavaScript und WebAssembly (WASM), um den Vergleich der Ausführungsgeschwindigkeiten zu ermöglichen. Die Trennung zwischen dem Backtester und den Strategien ermöglicht es das System zukünftig einfach, um weitere Sprachen und Schnittstellen zu erweitern.
Um möglichst viele Programmiersprachen unterstützten zu können, wird der Aufruf der Strategien über eine einfache Funktionsschnittstelle realisiert. Da WebAssembly nur Funktionsaufrufe erlaubt, eignet sich dieser Ansatz besonders gut, um eine breite Kompatibilität zu ermöglichen. In WebAssembly können keine Klassen nach außen definiert werden, weshalb die Funktionsschnittstelle die einzige Möglichkeit darstellt, mit dem Backtester zu kommunizieren. [45] [23]
Desweitern bietet die Funktionsschnittstelle den Vorteil, dass keine komplexen Objekte oder Datenstrukturen wie Klassen erzeugt werden müssen. Der Aufruf über eine Funktion ist im Vergleich zu objektorientierten Ansätzen in der Regel effizienter, vor allem bei Sprachenübergreifenden Schnittstellen. Dies vereinfacht die Implementierung der Strategien in verschiedenen Sprachen, verringert die Ausführungszeit und reduziert den Aufwand für die Integration in den Backtester. [28] [23]

3.2 Aufbau der Teststrategien

Zum Testen des Backtesters wurden verschiedene Konzepte für Handelsstrategien in konkrete Implementierungen überführt. Jeder der Strategieansätze verfolgt einen anderen Ansatz zur Generierung von Kauf- und Verkaufssignalen und weist dementsprechend auch unterschiedliche Geschwindigkeiten in der Ausführung auf. Der Aufbau der drei umgesetzten Strategien ist in den folgenden Unterkapiteln beschrieben.

3.2.1 Struktur der Mittelwertrückkehr Strategie mit Bollinger Bändern

Die Durchschnittsrückkehr beschreibt die Tendenz von Kurswerten, nach Abweichungen wieder in die Nähe ihres Mittelwerts zurückzukehren. Die Verwendung von Bollinger Bändern reduziert die Häufigkeit schwacher Handelssignale, indem kurzfristige Schwankungen herausfiltert werden und nur starke Preisabweichungen Impulse auslösen. Eine detaillierte Beschreibung dieser Strategieidee befindet sich in Abschnitt 2.4.1.
Als Breite des Bollinger Bandes wird eine Standardabweichung von 3,25 gewählt, um die Strategie an längerfristige Anlagen und Märkte mit höherer Volatilität anzupassen. Laut [32] sind Breiten von 1,8 Standartabweichungen von Bändern bei kurzzeitigen Investments vorteilhaft. Je mehr Schwankungen im Kursverlauf enthalten sind und bei einem längeren Anlagehorizont sollte eine größere Standardabweichung gewählt werden, um die Breite des Bandes entsprechend anzupassen. Somit werden mehr kurzfristige Signale entfernt und das Risiko für Fehlsignale gesenkt. Die Strategie löst erst dann Trades aus, wenn genügend Datenpunkte übergeben werden, da für die Berechnung der Standardabweichung eine Reihe von Werten benötigt wird. Die Mindestanzahl an historischen Kurswerten wird in der Strategie selbst festgelegt.

3.2.2 Aufbau der Ausbruchsstrategie

Ausbruchsstrategien erzeugen ein Kauf- oder Verkaufssignal, wenn der aktuelle Kurswert das in einem beobachteten Intervall ermittelte Maximum überschreitet oder das Minimum unterschreitet. Eine genauere Erklärung dazu befindet sich in dem Kapitel 2.4.3.
Die für diese Arbeit entwickelte Ausbruchsstrategie bezieht in die Minima und Maximum Berechnung alle erhaltenen Wert bis auf den Letzten mit ein. Der letzte Kurswert wird jedoch nicht direkt mit den Extremwerten des vergangenen Beobachtungszeitraums verglichen, sondern um geringfügige Kursschwankungen, die keinen tatsächlichen Ausbruchstrend anzeigen, zu vermeiden, mit einem Epsilon-Faktor angepasst. Dadurch sollen Fehlsignale in Seitwärtsmarktphasen vermieden werden. Ebenfalls zur Absicherung gegen Fehlsignale werden Haltesignale erst dann ausgegeben, wenn der Strategie mindestens die eingegebene Anzahl an Kurswerten übergeben wurde.

3.2.3 Umsetzung der gleitenden Durchschnittsüberschneidungs Strategie

Bei der Durchschnittsüberschneidungsstrategie werden zwei gleitende Durchschnitte mit unterschiedlich langen Zeiträumen berechnet und miteinander verglichen, um Handelssignale zu erzeugen. Wie bereits in Kapitel 2.4.2 beschrieben.
Die implementierte Handelsstrategie verwendet dabei einen gleitenden Durchschnitt mit einer Länge von acht Zeiteinheiten als kurzfristigen Durchschnitt, während für den langfristigen Durchschnitt ein Zeitraum von 24 Zeiteinheiten beobachtet wird. Dieses 1:3 Verhältnis wurde so gewählt, da es eine gute Balance zwischen Reaktionsfähigkeit, Gewinn und Stabilität bietet. [12] Diese Strategie berücksichtigt nur die aktuellsten 24 Datenpunkte, um die Durchschnitte zu berechnen. Alle übergebenen historischen Datenpunkte, die älter als dieses Intervall sind, werden ignoriert. Die Strategie generiert ausschließlich Haltesignale, sobald eine ausreichende Anzahl an Datenpunkten zur Berechnung der gleitenden Durchschnitte vorliegt. Zudem werden Handelssignale nur dann

ausgelöst, wenn die Überschneidung der Durchschnitte einen definierten prozentualen Schwellenwert in Relation zum aktuellen Kurs überschreitet, um die Entstehung von Fehlsignalen zu reduzieren.

3.3 Weboberfläche des Backtesters

Die Weboberfläche des Backtesters ermöglicht es, historische Kursdaten und Strategien hochzuladen, Backtests zu starten und die Ergebnisse zu visualisieren. Der Backtester ist nur über die Weboberfläche bedienbar. Die Webseite ist in verschiedene Bereiche unterteilt, die jeweils unterschiedliche Funktionen bereitstellen.

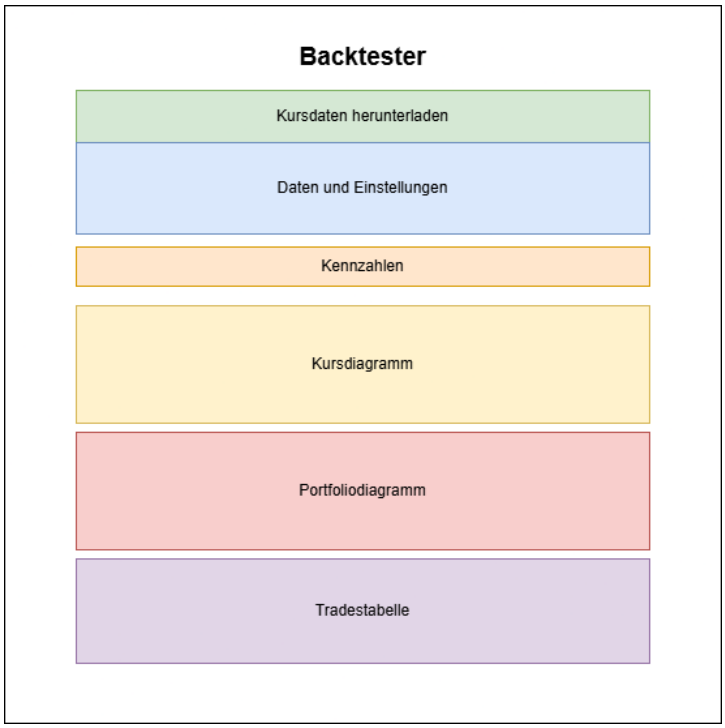


Abbildung 3.4: Design der Weboberfläche des Backtesters

In der Abbildung 3.4 ist das Design der Weboberfläche des Backtesters dargestellt. Der oberste Bereich bietet die Möglichkeit, historische Kursdaten im JSON Format herunterzuladen. Dabei muss das Kürzel des gewünschten Wertpapiers sowie der Zeitraum angegeben werden. Unterhalb befindet sich der Abschnitt, in dem die Kursdaten und die Handelsstrategie hochgeladen sowie die Startparameter für den Backtest festgelegt werden können. Zu diesen Parametern zählen das Anfangskapital, das Intervall, in dem die Strategie abgefragt werden soll, sowie die Anzahl der historischen Datenpunkte, die an die Strategie übergeben werden. Nachdem alle Felder ausgefüllt und als gültig erkannt wurden, kann der Backtest per Knopfdruck gestartet werden. Im folgenden Bereich werden nach der Berechnung die Kennzahlen des Backtests dargestellt. Daraufhin werden die Kursverläufe mit den Handelszeitpunkten sowie die Portfolioentwicklung visualisiert. Abschließend wird eine Tabelle mit allen im Rahmen des Backtests ausgeführten

Trades angezeigt.

3.4 Visualisierung der Daten und Ergebnisse

Die Kennzahlen, Ergebnisse des Backtestes und Kursdaten werden als Liniendiagramme visualisiert. Dabei werden diese in zwei Diagrammen dargestellt. 3.5 Das obere Diagramm zeigt die Kursdaten über die Zeit an und markiert an welchen Stellen ver- oder gekauft wurde. Das untere Diagramm stellt den Portfoliowert über die Zeit dar. Das Portfoliodiagramm und die Kennzahlen helfen die Effektivität der getesteten Strategie zu einzuschätzen und machen die Performance visuell sichtbar.

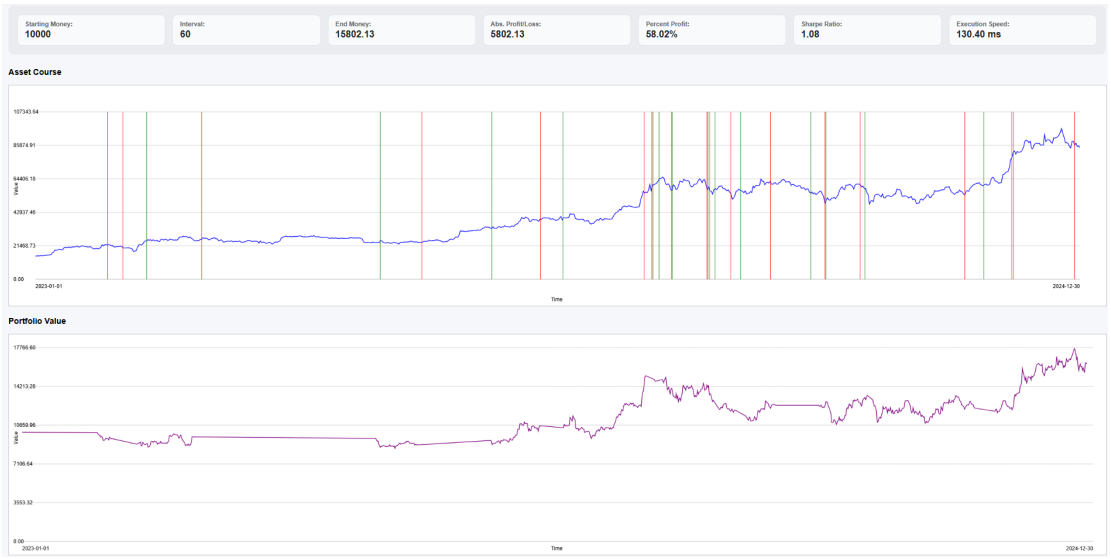


Abbildung 3.5: Kursdiagramme mit Kennzahlen

Bei der Visualisierung der Finanzdaten fallen große Datenmengen mit Millionen von Datenpunkten an. Um eine übersichtliche Darstellung zu ermöglichen, werden die Datensätze daher zur Darstellung durch Downsampling reduziert. Zur Reduktion des Datensets wird eine stichprobenartige Reduktion mittels Schrittweite verwendet, bei der jeder n-te Datenpunkt berücksichtigt wird. Dadurch wird die Anzahl der dargestellten Datenpunkte verringert, während die grundlegende Form des Aktienkurses erhalten bleibt. Durch die Wahl der Schrittweite kann die Detailliertheit der Darstellung angepasst werden. [34]

3.5 Aufbau der Geschwindigkeitstests

Zur Evaluierung der Ausführungsgeschwindigkeit der implementierten Strategien in den verschiedenen Programmiersprachen wird bei jedem Backtest die Laufzeit erfasst. Da der Fokus auf dem Vergleich der Ausführungszeiten zwischen den Programmiersprachen liegt, wird die Zeitmessung bei mehreren identen Strategielogiken in verschiedenen Sprachen durchgeführt. Als Ausführungsumgebungen werden die Webbrowser Google

Chrome und FireFox verwendet. Google Chrome ist mit etwa 70 % Marktanteil, der am häufigsten genutzte Browser und bietet somit eine realistische Einschätzung der Performance. [40, 44] FireFox wird als zweiter Testbrowser eingesetzt, da dieser die schnelleste Ausführungsgeschwindigkeit für WebAssembly Module bietet und somit einen interessanten Vergleichspunkt darstellt, wie in Kapital 2.5.1 beschrieben.
Die Backtests werden mit unterschiedlich großen Datensätzen durchgeführt, um die Performance der Strategien bei variierenden Datenmengen zu bewerten. Ebenfalls werden alle Tests auf derselben Hardware ausgeführt, um eine Vergleichbarkeit der Ergebnisse zu gewährleisten. Zusätzlich wird jede Strategie mehrfach auf demselben Datensatz angewendet, um die gemessenen Laufzeiten zu Mitteln und zufällige Ausreißer durch Hintergrundprozesse oder Just-in-Time-Kompilierung zu reduzieren. Die gemessenen Ausführungszeiten werden in Millisekunden gemessen und am Ende jedes Backtestes angezeigt.

3.6 Abrufen der historischen Kursdaten

Die historischen Kursdaten werden über Binance bezogen. [10] Binance bietet eine API an, über die Finanzdaten für verschiedene Aktien oder Kryptowährungen in gewissen Zeiträumen abgerufen werden können. Die Daten werden aber nicht direkt weiterverwendet, sondern in einem Vorverarbeitungsschritt in das benötigte JSON Format umgewandelt. Dabei wird der durchschnittliche Kurswert pro Minutenintervall berechnet und zusammen mit dem Zeitstempel in die JSON Datei geschrieben. Mit der Normalisierung der Daten auf Minuten wird eine einheitliche Datenbasis für die Backtests geschaffen werden. Die Datei kann anschließend lokal gespeichert werden. Die verarbeiten Datensets können in der Weboberfläche des Backtesters hochgeladen und verwendet werden.

3.7 Werkzeuge zur WASM Entwicklung

Für die Entwicklung von WebAssembly (WASM) können verschiedenste Werkzeuge eingesetzt werden. Ein Compiler ist notwendig, um den Quellcode in das WASM-Format zu übersetzen. Häufig verwendete Compiler sind dafür Emscripten für C/C++ [16, 52], TeaVM für Java [2, 49] oder rustc für Rust [17].
Des Weiteren können Tools wie das The WebAssembly Binary Toolkit (WABT) [53] zur Optimierung und Analyse von WASM-Modulen verwendet werden.
In dieser Arbeit wurde Emscripten als Compiler von C Code und WABT als Tool zur Auswertung von kompilierten WASM Dateien eingesetzt.

3.7.1 Emscripten

Emscripten ist ein Open Source Compiler, der C und C++ Code in WebAssembly (WASM) übersetzt.

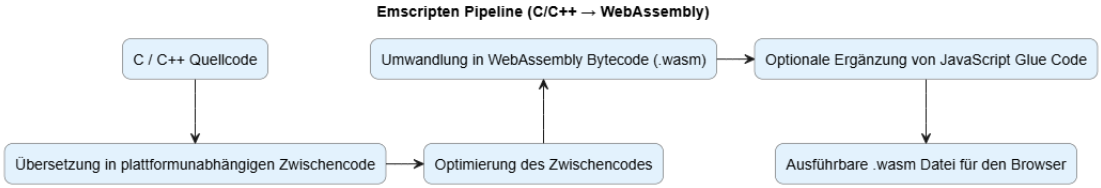


Abbildung 3.6: Emscripten Pipeline

In der Abbildung 3.6 ist der Ablauf eines Emscripten Kompiliervorgangs dargestellt. Zuerst wird der C/C++ Quellcode mit dem Clang Frontend in eine Zwischensprache übersetzt. Die Zwischensprache wird danach je nach Compileroption optimiert, um die Performance zu verbessern. Im nächsten Schritt entsteht aus der optimierte Zwischensprache WebAssembly Code. Abschließend kann optional JavaScript-Glue-Code generiert werden, der das Aufrufen des WebAssembly Moduls in JavaScript erleichtert. Dieser zusätzliche Code kann jedoch die Ausführungszeit des Moduls beeinträchtigen. [15]
Um die bestmögliche Performance zu erzielen, wird in dieser Arbeit auf die Generierung von JavaScript-Glue-Code verzichtet und mit der höchsten Optimierungsstufe O3 kompiliert. Somit wird gewährleistet, dass der erzeugte WebAssembly Code möglichst effizient ausgeführt werden kann.
Alle Strategien sind wie folgt kompiliert:
emcc strategie.c -O3 -nostdlib -Wl,–no-entry -Wl,–allow-undefined -o strategie.wasm

- **emcc**: Befehl zum Aufrufen von Emscripten.
- **strategie.c**: Die C-Quellcodedatei der jeweiligen Strategie.
- **-O3**: Die Optimierungsstufe.
- **-nostdlib**: Verhindert das Einbinden von Standardbibliotheken.
- **-Wl,–no-entry**: Gibt an, dass das Modul keinen Einstiegspunkt hat. In diesem Fall ist dies die fehlende main Funktion, da es sich um ein Modul handelt.
- **-Wl,–allow-undefined**: Erlaubt undefinierte Symbole im Modul.
- **-o strategie.wasm**: Legt den Namen der Ausgabedatei fest.

3.7.2 WABT

Das WebAssembly Binary Toolkit (WABT) bietet eine Zusammenstellung von Werkzeugen zur Untersuchung und nachträglichen Änderung von WebAssembly (WASM) Modulen. WASM Dateien können damit beispielsweise in ein menschenlesbares Textformat umgewandelt, dekompiliert oder nachträglich verändert und validiert werden. Im lesbaren Textformat ist der Bytecode nicht mehr binär kodiert, sondern durch Assembler Code dargestellt, der leichter zu interpretieren ist. WABT bietet des weiteren Werkzeuge zur Darstellung in JSON oder einem Object Dump Format an. Ebenfalls ist es möglich Metainformationen über das Modul zu extrahieren, wie beispielsweise die Anzahl der Funktionen, den Speicherverbrauch oder die importierten und exportierten Symbole. [53] Der in der Abbildung gezeigte Bytecode wurde mit WABT decodiert. Eine Darstellung des Bytecodes ist in der Abbildung 2.4 zu sehen.

Kapitel 4

Implementierung des Backtesters

In diesem Kapitel wird beschrieben wie die einzelnen Komponenten des Backtester implementiert sind.

4.1 Die Benutzerschnittstellen

4.1.1 Die API Schnittstelle

4.1.2 Die Logikeingangsparameter

4.2 Implementierung der Backtestlogik

4.3 Implementierungen der Strategien

4.3.1 Gleitende Durchschnittsüberschneidungs Strategie

4.3.2 Mittelwertrückkehr Strategie

4.3.3 Ausbruchsstrategie

4.4 Die Datenvisualisierungen

4.4.1 Visualisierung des Kurswertes

4.4.2 Visualisierung des Portfoliowertes

Kapitel 5

Performancetests der unterschiedlichen Programmiersprachen

In diesem Kapitel werden die Rahmenbedingungen der Tests beschrieben, die durchgeführten Performancetests analysiert und die Ergebnisse interpretiert.

5.1 Die Testvoraussetzungen

Damit die Ergebnisse der Performancetests miteinander vergleichen zu können, müssen die Tests unter den gleichen Voraussetzungen durchgeführt werden. Deswegen wird in diesem Abschnitt beschrieben, welche Voraussetzungen für die Tests geschaffen wurden.

5.1.1 Die Testumgebung

5.1.2 Die Testdaten

Alle Strategien, die in den Performancetests getestet werden, verwenden die gleichen historischen Kursdatensätze. Für die Tests werden Kursdaten von drei verschiedenen Kryptomünzen verwendet:

- Bitcoin (BTC)
- Ethereum (ETH)
- Cardano (ADA)

Diese Kursdaten werden von der Kryptobörse Binance [10] bezogen und umfassen jeweils einen Zeitraum von ein, zwei und drei Jahren. Für die Testfälle wird Bitcoin im Zeitraum vom 01.07.2023 bis zum 30.06.2025, Cardano im Zeitraum vom 01.01.2021 bis zum 31.12.2022 und Ethereum im Zeitraum vom 01.01.2023 bis zum 31.12.2023 betrachtet. Die Kursdaten umfassen etwa 500.000 Datenpunkte pro Jahr und liegen in einem Minutenintervall vor. Somit stehen für Bitcoin rund 1,5 Millionen, für Cardano etwa 1 Million und für Ethereum etwa 500.000 Kurswerte zur Verfügung. Durch die Kombination der Beobachtungszeiträume und Kryptowährungen wird ein breites Spektrum an unterschiedlichen Kursbewegungen abgebildet, sodass verschiedene Marktphasen und stark variierende Datenpunktumfänge in den Tests berücksichtig werden.

5.2 Die Testausführung

5.3 Die Testergebnisse

5.3.1 Ergebnisse der gleitenden Durchschnittsüberschneidungs Strategie

5.3.2 Ergebnisse der mittelwertrückkehr Strategie

5.3.3 Ergebnisse der Ausbruchsstrategie

5.4 Analyse der Testergebnisse

Kapitel 6

Zusammenfassung und Schlussbemerkungen

2 Seiten diese und nächste

Quellenverzeichnis

Literatur

[1] Madhav Agarwal. *Breakout Pattern - Meaning and Trading Strategy*. Accessed: 2025-09-19. 2024. URL: https://www.wrightresearch.in/blog/breakout-pattern-meaning-and-trading-strategy/ (siehe S. 9).

[2] Alexey Andreev und TeaVM Contributors. *TeaVM Website*. Accessed: 2025-10-26. 2013. URL: https://teavm.org/ (siehe S. 19).

[3] P. Arumugam und R. Saranya. „An Empirical Analysis of Trading Strategy Based on Simple Moving Average Crossovers“. *ICTACT Journal on Management Studies* 3.1 (2017), S. 423–426 (siehe S. 9).

[4] David H. Bailey und Marcos López de Prado. „How “Backtest Overfitting” in Finance Leads to False Discoveries“. *Significance* 18.6 (2021), S. 22–25 (siehe S. 3).

[5] David H. Bailey u. a. „Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance“. *Notices of the American Mathematical Society* 61.5 (2014), S. 458–471 (siehe S. 4).

[6] David H. Bailey u. a. „The Probability of Backtest Overfitting“. *Journal of Computational Finance (Risk Journals)* (2015) (siehe S. 2, 4).

[7] Andrew Baronick. *BacktestJS Example Result*. Accessed: 2025-09-29. 2024. URL: https://backtestjs.com/docs/trading-strategy-results/view-trading-strategy-results/ (siehe S. 7).

[8] Andrew Baronick. *BacktestJS Website*. Accessed: 2025-09-29. 2024. URL: https://backtestjs.com/ (siehe S. 7).

[9] Rim El Bernoussi und Michael Rockinger. „Rebalancing with transaction costs: theory, simulations, and actual data“. *Financial Markets and Portfolio Management* 37.2 (2023), S. 121–160 (siehe S. 4).

[10] Binance. *Binance - Cryptocurrency Exchange*. https://www.binance.com/. Accessed: 2025-11-03. 2025 (siehe S. 19, 22).

[11] John Bollinger. „Using bollinger bands“. *Stocks & Commodities* 10.2 (1992), S. 47–51 (siehe S. 8).

[12] WILLIAM BROCK, JOSEF LAKONISHOK und BLAKE LeBARON. „Simple Technical Trading Rules and the Stochastic Properties of Stock Returns“. *The Journal of Finance* 47.5 (1992), S. 1731–1764 (siehe S. 9, 16).

[13] Stephen J. Brown u. a. „Survivorship Bias in Performance Studies“. *The Review of Financial Studies* 5.4 (2015), S. 553–580 (siehe S. 5).

[14] Mark M. Carhart u. a. „Mutual Fund Survivorship“. *The Review of Financial Studies* 15.5 (2015), S. 1439–1463 (siehe S. 5).

[15] Emscripten Contributors. *Emscripten Toolchain Website*. Accessed: 2025-10-26. 2015. URL: https://emscripten.org/ (siehe S. 20).

[16] Emscripten Contributors. *Emscripten Website*. Accessed: 2025-10-26. 2015. URL: https://emscripten.org/ (siehe S. 19).

[17] Mozilla.org contributors. *Compiling from Rust to WebAssembly*. Accessed: 2025-10-26. 2025. URL: https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Rust_to_Wasm (siehe S. 19).

[18] Gilles Daniel, Didier Sornette und Peter Wohrmann. *Look-Ahead Benchmark Bias in Portfolio Performance Evaluation*. 2008 (siehe S. 4).

[19] Poltavskyi Dmytro. „Integration of WebAssembly in Performance-critical Web Applications“. *American Scientific Research Journal for Engineering, Technology, and Sciences* 102.1 (2025), S. 39–53 (siehe S. 11).

[20] Fernando F. Ferreira, A. Christian Silva und Ju-Yi Yen. „Detailed Study of a Moving Average Trading Rule“. *Quantitative Finance* 18.9 (2018), S. 1599–1617 (siehe S. 9).

[21] Luis A. Gil-Alana. „Mean reversion in the real exchange rates“. *Economics Letters* 69.3 (2000), S. 285–288 (siehe S. 8).

[22] Ikhlaas Gurrib. „The Moving Average Crossover Strategy: Does It Work for the S&P500 Market Index?“ *Global Review of Accounting and Finance* 7.1 (2016), S. 92–107 (siehe S. 9).

[23] Andreas Haas u. a. „Bringing the web up to speed with WebAssembly“. *SIGPLAN Not.* 52.6 (2017), S. 185–200 (siehe S. 15).

[24] Campbell Harvey und Yan Liu. „Backtesting“. *SSRN Electronic Journal* 42 (2013) (siehe S. 3).

[25] David Herrera u. a. „Numerical Computing on the Web: Benchmarking for the Future“. In: *Proceedings of the 14th ACM SIGPLAN International Symposium on Dynamic Languages (DLS 2018)*. ACM, 2018, S. 88–100 (siehe S. 11).

[26] Ulf Holmberg, Carl Lönnbark und Christian Lundström. „Assessing the profitability of intraday opening range breakout strategies“. *Finance Research Letters* 10.1 (2013), S. 27–33 (siehe S. 9).

[27] Jing-Zhi Huang und Zhijian (James) Huang. „Testing moving average trading strategies on ETFs“. *Journal of Empirical Finance* 57 (2020), S. 16–32 (siehe S. 9).

[28] John Hughes. „Why Functional Programming Matters“. In: *Research Topics in Functional Programming*. Hrsg. von David Turner. Addison-Wesley, 1990, S. 17–42 (siehe S. 15).

[29] Quantopian Inc. *Zipline Trader Website*. Accessed: 2025-09-30. 2020. URL: https://zipline-trader.readthedocs.io/en/latest/index.html (siehe S. 6).

[30] Quantopian Inc. und Stefan Jansen. *Zipline Reloaded Website*. Accessed: 2025-09-30. 2020. URL: https://zipline.ml4trading.io/ (siehe S. 6).

[31] Abhinav Jangda u. a. „Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code“. In: *2019 USENIX Annual Technical Conference (USENIX ATC '19)*. USENIX Association, 2019, S. 107–120 (siehe S. 11).

[32] Audrius Kabasinskas und Ugnius Macys. „Calibration of Bollinger Bands Parameters for Trading Strategy Development in the Baltic Stock Market“. *Inzinerine Ekonomika-Engineering Economics* 21.3 (2010), S. 244–254 (siehe S. 16).

[33] Mark Leeds. *Bollinger Bands Thirty Years Later*. 2013 (siehe S. 8).

[34] Jure Leskovec und Christos Faloutsos. „Sampling from large graphs“. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. Philadelphia, PA, USA: Association for Computing Machinery, 2006, S. 631–636 (siehe S. 18).

[35] Andrew W. Lo und A. Craig MacKinlay. *Data-snooping Biases in Tests of Financial Asset Pricing Models*. Techn. Ber. No. 3020-89-EFA. Massachusetts Institute of Technology, Sloan School of Management, 1989 (siehe S. 5).

[36] Richard Martin und Torsten Schöneborn. *Mean Reversion Pays, but Costs*. 2011 (siehe S. 8).

[37] Cory Mitchell. *Failed Break: What It Is, How It Works, Example*. Accessed: 2025-09-19. 2023. URL: https://www.investopedia.com/terms/f/failedbreak.asp (siehe S. 9).

[38] James M. Poterba und Lawrence H. Summers. „Mean reversion in stock prices: Evidence and Implications“. *Journal of Financial Economics* 22.1 (1988), S. 27–59 (siehe S. 8).

[39] Daniel Rodriguez. *Backtrader Website*. Accessed: 2025-09-29. 2015. URL: https://www.backtrader.com/ (siehe S. 6).

[40] StatCounter Global Stats. *Browser Market Share Worldwide*. https://gs.statcounter.com/browser-market-share. Accessed: 2025-10-31. 2025 (siehe S. 19).

[41] Guanru Su. „Analysis of the Bollinger Band Mean Regression Trading Strategy“. In: *Proceedings of the 3rd International Conference on Economic Development and Business Culture (ICEDBC 2023)*. Atlantis Press, 2023, S. 95–102 (siehe S. 8).

[42] Ryan Sullivan, Allan Timmermann und Halbert White. „Data-Snooping, Technical Trading Rule Performance, and the Bootstrap“. *The Journal of Finance* 54.5 (1999), S. 1647–1691 (siehe S. 5).

[43] Mircea Ţălu. „A Comparative Study of WebAssembly Runtimes: Performance Metrics, Integration Challenges, Application Domains, and Security Features“. *Archives of Advanced Engineering Science* 00.00 (2025), S. 1–13 (siehe S. 11).

[44] W3Counter. *Browser & Platform Market Share – Global Web Stats*. https://www.w3counter.com/globalstats.php. Accessed: 2025-10-31. 2025 (siehe S. 19).

[45] WebAssembly Community Group. *WebAssembly*. Accessed: 2025-09-07. 2025. URL: https://webassembly.org/ (siehe S. 10, 15).

[46] Yutian Yan u. a. „Understanding the performance of webassembly applications". In: *Proceedings of the 21st ACM Internet Measurement Conference*. Association for Computing Machinery, 2021, S. 533–549 (siehe S. 11).

[47] Carlo Zarattini, Andrea Barbon und Andrew Aziz. *A Profitable Day Trading Strategy For The U.S. Equity Market*. Research Paper 24-98. Swiss Finance Institute, 2024 (siehe S. 9).

[48] Saurabh Zunke und Veronica D'Souza. „JSON vs XML: A Comparative Performance Analysis of Data Exchange Formats". *International Journal of Computer Science and Network* 3.4 (2014), S. 257–261 (siehe S. 13).

Software

[49] Alexey Andreev und TeaVM Contributors. *TeaVM GitHub*. Accessed: 2025-10-26. 2013. URL: https://github.com/konsoletyper/teavm (siehe S. 19).

[50] founder) BacktestingMax (Max. *BacktestingMax*. Accessed: 2025-10-18. 2025. URL: https://backtestingmax.com (siehe S. 5).

[51] Andrew Baronick. *BacktestJS GitHub*. Version 1.0.0. Accessed: 2025-09-29. 2024. URL: https://github.com/andrewbaronick/BacktestJS (siehe S. 7).

[52] Emscripten Contributors. *Emscripten GitHub*. Accessed: 2025-10-26. 2015. URL: https://github.com/emscripten-core/emsdk (siehe S. 19).

[53] WebAssembly Community Group. *WebAssembly Binary Toolkit (WABT) GitHub*. Accessed: 2025-10-26. 2025. URL: https://github.com/WebAssembly/wabt (siehe S. 19, 20).

[54] Quantopian Inc. *Quantopian PyFolio GitHub*. Version 0.9.2. Accessed: 2025-09-30. 2015. URL: https://github.com/quantopian/pyfolio (siehe S. 6).

[55] Quantopian Inc. *Zipline GitHub*. Version 1.4.1. Accessed: 2025-09-30. 2020. URL: https://github.com/quantopian/zipline (siehe S. 6).

[56] Quantopian Inc. und Stefan Jansen. *Zipline Reloaded GitHub*. Version 3.1.1. Accessed: 2025-09-30. 2020. URL: https://github.com/stefan-jansen/zipline-reloaded (siehe S. 6).

[57] Quantopian Inc. und Shlomi Kushchi. *Zipline GitHub*. Version 1.6.0. Accessed: 2025-09-30. 2020. URL: https://github.com/shlomiku/zipline-trader (siehe S. 6).

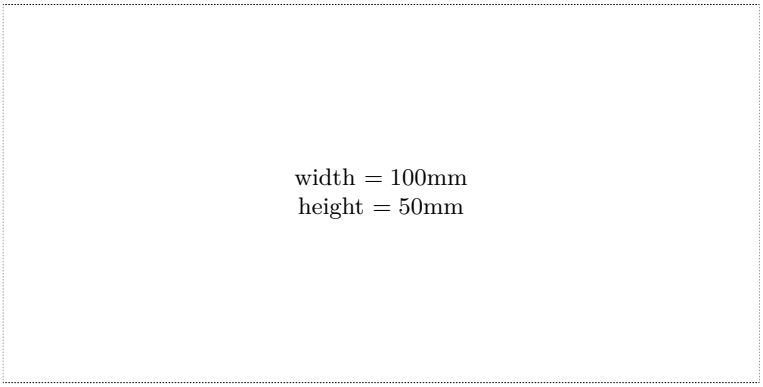
[58] Daniel Rodriguez. *Backtrader GitHub*. Version 1.9.78.123. Accessed: 2025-09-29. 2015. URL: https://github.com/mementum/backtrader (siehe S. 6).

[59] Backtester.io team. *Backtester.io*. Accessed: 2025-10-18. 2025. URL: https://backtester.io (siehe S. 5).

[60] FX Replay team. *FX Replay*. Accessed: 2025-10-18. 2025. URL: https://www.fxreplay.com/ (siehe S. 5).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —